

# Deterministic Near-Optimal P2P Streaming

Shaileshh Bojja Venkatakrisnan\*, Pramod Viswanath<sup>†</sup>

*Coordinated Science Laboratory*

*University of Illinois - Urbana Champaign, USA*

*Email: {<sup>\*</sup>bjjvnkt2, <sup>†</sup>pramodv}@illinois.edu*

## Abstract

We consider streaming over a peer-to-peer network with homogeneous nodes in which a single source broadcasts a data stream to all the users in the system. Peers are allowed to enter or leave the system (adversarially) arbitrarily. Previous approaches for streaming in this setting have either used randomized distribution graphs or structured trees with randomized maintenance algorithms. Randomized graphs handle peer churn well but have poor connectivity guarantees, while structured trees have good connectivity but have proven hard to maintain under peer churn. We improve upon both approaches by presenting a novel distribution structure with a *deterministic* and distributed algorithm for maintenance under peer churn; our result is inspired by a recent work [1] proposing deterministic algorithms for rumor spreading in graphs.

A key innovation in our approach is in having *redundant* links in the distribution structure. While this leads to a reduction in the maximum streaming rate possible, we show that for the amount of redundancy used, the delay guarantee of the proposed algorithm is near optimal. We introduce a *tolerance* parameter that captures the worst-case transient streaming rate received by the peers during churn events and characterize the fundamental tradeoff between rate, delay and tolerance. A natural generalization of the deterministic algorithm achieves this tradeoff near optimally. Finally, the proposed deterministic algorithm is robust enough to handle various generalizations: ability to deal with heterogeneous node capacities of the peers and more complicated streaming patterns where multiple source transmissions are present.

## 1. Introduction

In peer-to-peer (P2P) streaming, a low-capacity server uploads the content to a small number of clients which, together with the other clients (a total of  $n$  peers), then exchange the content among themselves. This is similar to the rumor spreading problem, in which a rumor from a source node is propagated to all the nodes of an unknown network. However unlike rumor spreading, where only a single rumor is communicated to neighbors over many rounds, in streaming new “rumors” arrive continuously in an online fashion and need to be forwarded fast and effectively in order to prevent message loss. Limited upload capacity of peers disallows flooding-type message forwarding. Further, peers can arrive or depart the system at will (peer churn), requiring scheduling algorithms to be designed in order to effectively utilize the upload capacity available and to ensure playback continuity with small delay.

In this work, we consider the problem of constructing and maintaining a P2P overlay network  $G(V, E)$  in a distributed fashion subject to the following restrictions. Peers can contact other peers, if they know their addresses, and form data carrying communication links on  $G(V, E)$ . While new addresses can be learnt by the peers by talking to their neighbors in  $G(V, E)$ , peers have a constant bound on (i) the number of addresses they can remember at any time, (ii) out-degree and (iii) upload capacity. Also, peers have only local knowledge of the topology of the graph  $G(V, E)$ . A server node receives data packets continuously as a live-stream from a source external to the network. The problem now is to construct  $G(V, E)$  in order to distribute the data-stream from the source to all the peers. Additionally, we want to distribute the packets as quickly as possible (delay) and as many as possible in any given time duration (rate). As such there exist many algorithms that can stream optimally in this setting [2], [3]. However, in practical P2P systems [4], peers seldom stay in the network all the time. To model this we let the nodes enter or exit the system arbitrarily. With this additional assumption on peer dynamics, repairing  $G(V, E)$  to maintain fresh flow of packets to the peers, and ensuring good delay at the same time, becomes a challenging issue. In particular, we consider a setting where the number of simultaneous connected departures is bounded (see section 2) but require that the peers may suffer a loss of at most a constant number of packets after each round of departure. For example, if any one peer departs the system then the remaining peers can experience a rate loss for at most a constant number of rounds before continuing to receive the full rate as before.

A popular method used by some early systems, was to divide the content into multiple substreams and distribute via *multicast trees* having disjoint interior nodes [2], [3], [5]–[7]. This way any peer could be an interior node in one multicast tree where it utilizes its upload bandwidth. While trees offer good playback rate and delay, managing trees in a distributed fashion can be very difficult under peer churn. It is known that the complexity of maintaining trees grows with the number of nodes [8], [9]. Hence, random sampling by the peers has commonly been used to help maintain the distribution trees [10]. Another line of work introduced randomness in the construction of the distribution graphs themselves in order to handle the problems associated with peer churn [11]. Whenever a neighboring peer leaves, the peer chooses a new neighbor randomly as its new neighbor. While the distributed nature of the peer pairing makes unstructured networks robust to peer churn, connectivity is sacrificed because some of the peers may not be well connected due to the inherent randomness.

Thus, while structured algorithms promise connectivity to all the peers and have deterministic  $O(\log n)$  delay guarantees, a fundamental limitation is their vulnerability to peer churn. Randomized algorithms, on the other hand, provide only probabilistic guarantees for delay, convergence time and connectivity guarantees are weaker. Besides few algorithms (an exception is [11]) provide a formal guarantee on the transient rates received by the peers. A similar trend can be found in the literature on gossip, where a long line of algorithms tried to reduce the spreading time for randomized gossip [12]–[14]. However, recently a deterministic distributed algorithm for gossip was proposed in [1]. Apart from being faster and more robust than previous randomized algorithms, the deterministic nature has the advantage of running

time guarantees holding with certainty instead of with high probability. Inspired by this, we propose and analyze a novel distribution structure for P2P streaming that can be maintained deterministically, distributedly by the peers and provides a strong transient rate guarantee under our departure model.

## 1.1. Our Results

The foremost challenge for the streaming problem in the setting discussed above is the design of the algorithm to construct and maintain the P2P overlay. This is in contrast to the literature on gossip wherein the model and algorithms, to some extent (such as uniform random gossip [13], [14]), are fairly standard and much of the innovation is in the analysis. Our main result is the design of the distribution structure and algorithm that is (i) distributed, (ii) deterministic and (iii) has constant repair time to ensure connectivity. As far as the authors are aware, no other algorithm in the literature has all of the above properties. The key innovation is the introduction of *redundancy* in the network. Assuming the peers have an upload capacity of  $C$  each, the delay provided by our algorithm is given by the following.

**Theorem 1.** *In the steady state if there are  $n$  peers in the system, the streaming delay is bounded by  $\log_2(n+1) + \frac{2R}{C-R} + \log_2(1 - \frac{R}{C}) - 2$  for a rate  $R \in (0, C)$ .*

The above delay of our algorithm has an additional term of  $O(1/(C-R))$  as compared to the  $O(\log n)$  delay of tree based structures, such as in [3], [10]. However, the latter tree based algorithms do not have constant repair time under churn. Peer departures can cause a sudden loss of transmission links and can lead to loss of connectivity in the multicast graphs. Under such events, the data rate received by some peers can drop considerably until the trees are repaired. Having redundancy in the network helps in this regard in ensuring continuity of streaming without outages under peer churn. The penalties paid due to the introduction of redundant links facilitate: (i) deterministic graph management and (ii) ensure continuity of playback under peer churn events. In our second result, we show that for the amount of redundancy used, the delay guarantee of the algorithm is order optimal.

**Theorem 2.** *For streaming using multiple structured graphs, each carrying partial flows, if each of the substream graphs have enough capacity redundancy to handle arbitrary node departures, then the maximum delay across the substream graphs is at least  $\log_\Delta(n) + \frac{R}{2(C-R)} + \log_\Delta\left(\frac{2(C-R)}{R}\right) - c$ , where  $c = (\Delta - 2) \log_\Delta\left(\frac{\Delta!}{2}\right) + \log_e(\Delta - 1) + 2$ , for a rate  $R$ , degree bound  $\Delta$  and  $n \geq \frac{3R}{C-R}$ .*

Thus, we claim that the  $R/(C-R)$  term in the delay is *fundamental* for all algorithms guaranteeing continuity of playback. We also guarantee a transient rate equal to the original rate under bounded departure events. The transient rate can be traded for delay as discussed in section 6. Hence apart from providing deterministic guarantees for delay and churn management, the algorithm offers key insights into the continuity aspect of the playback rate. It also extends readily for all-cast streaming, where every peer has a stream to be broadcast, and the case where the peers have heterogeneous upload capacities (Appendices C and D).

## 1.2. Related Work

A standard approach in structured streaming involved multicast trees, and often with constant-degree nodes [23]–[25]. Several approaches have been presented towards the management of the trees. Algorithms in [2], [5], [7] used centralized control. Pastry [17], a routing substrate, was used by the SplitStream algorithm in [3] for tree construction and maintenance. Other distributed lookup protocols have also been proposed in [15], [16]. In [10], an asynchronous distributed algorithm was presented to construct and manage multiple distribution trees by means of random sampling done by the peers. In the other research direction of unstructured P2P networks, where each node communicates with a random subset

Flow dissemination graph type	Graph maintenance algorithm	References
Structured	Centralized	[2], [5], [7]
	Involves randomness	[3], [6], [10], [15]–[17]
	Deterministic	<b>This paper</b>
Unstructured	Random	[11], [18]–[22]

Table 1. Summary of comparison with previous work for flow based streaming.

of other peers, much of the previous theoretical studies of the delay performance have focused primarily on fully connected networks with homogeneous capacities; examples include [18]–[20] which make interesting connections between P2P streaming networks and gossip and epidemic models to analyze the maximum streaming delay. In [11], multiple random Hamiltonian cycles were constructed and superposed. The distribution is then done over the union of the cycles. A key idea was to exploit the fact that the superposition of random directed Hamiltonian cycles is an expander with high probability. Additionally, Hamiltonian cycles are easy to maintain in response to peer churn, a fact that was first noted in the case of undirected graphs in [26]. Some other formats of unstructured P2P include mesh based streaming in [21], [22], in which packets were distributed over a randomly constructed mesh. A comparison between the previous work discussed above and this work has been presented in Table 1. We note that the idea of using redundancy to counter transient effects has been observed in other contexts as well [27], [28].

## 2. System Model

The P2P overlay network  $G(V(t), E(t))$ , where the time  $t$  is slotted, is assumed to be an undirected *node capacitated* graph in which all the peers have a uniform upload capacity of  $C$  and a constant bound  $\Delta$  on the number of upload links allowed. In addition to the upload capability, we let the nodes be able to communicate  $O(\log n_{\max}(t))$  bits of information in any round  $t$  as control messages through the edges where  $n_{\max}(t)$  denotes the maximum number of peers that were in the system up until time  $t$ . Peers have a constant amount of memory for storing  $M$  addresses (node ID’s) and are allowed to arrive and depart from the system arbitrarily. When a peer departs, the node and all the edges connected to it are lost immediately; only the neighbors of the departing peer(s) in  $G(V(t), E(t))$  are aware of this event. Let us call the maximally connected sets among the departing peers, in  $G(V(t), E(t))$ , as “peer departure blocks”. For example, if  $G(V(t), E(t))$  is as in Figure 2(a) and peers 3, 5 and 6 leave the system at the same time, then  $\{3\}$ ,  $\{5,6\}$  constitute the peer departure blocks. We assume that peer departure blocks, at any time, are of size at most  $K$ . Here  $K$  has a linear dependence on  $M$ . Peer arrivals, in which a new peer becomes part of the overlay, happens at most one at a time. We also assume communication happens as a flow (or equivalently as time-shared discrete messages) and do not consider network coding in this paper. Notation: for any positive integer  $n$ ,  $[n]$  denotes the set  $\{1, 2, \dots, n\}$ .

In the above model of peer dynamics, peers can potentially arrive or depart frequently. As such the P2P network can be constantly changing to adapt to that. Let us call such a state of the network, which is in the process of reconfiguring itself, as a *transient* state. We call a state where the network is no longer changing as a *steady state*. This can happen, for example, if the time gap following a churn event and until the next churn event is large. In the following section, we discuss the steady state network structure of our algorithm.

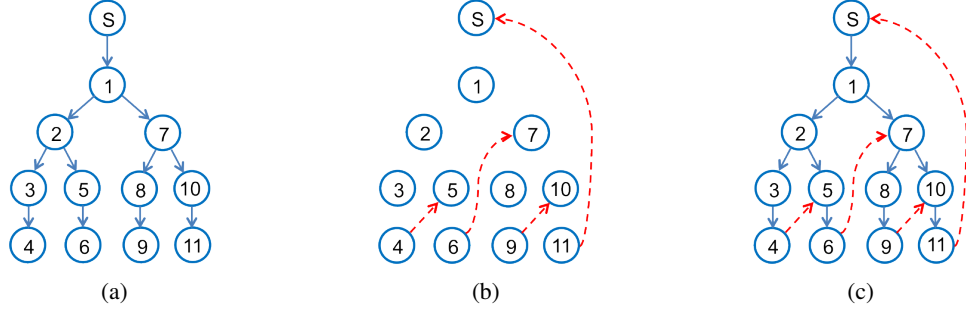


Figure 1. An example showing the directed graphs (a)  $T_1$ , (b)  $U_1$ , and (c)  $G_1$  for a network with  $n = 11$  peers and  $m = 3$ . The node indices correspond to the peer labels for  $G_1$  and node  $S$  is the stream source.

### 3. Overview of Steady State Structure

In the following sections, we let  $C = 1$  without loss of generality. We also let  $K = 1$  in the departure model. Let us consider a streaming of rate  $R \leq 1$  done over  $T$  distribution graphs. Each distribution graph then is used to disseminate a rate  $R/T$  substream to the peers. Let  $G_i(V(t), E_i(t))$ , for  $i = 1, \dots, T$ , denote the directed graph for broadcasting the  $i$ th substream, where  $V(t)$  and  $E_i(t)$  denote the set of all users in the system and the set of links used for the  $i$ th substream at time  $t$  respectively. Each user is interested in receiving all the  $m$  substreams (we do not assume any coding done over the data stream such as Multiple Description Coding (MDC) [3]). For ease of notation, we will drop the argument  $t$  from  $G_i(t), E_i(t)$  etc. and denote them simply by  $G_i, E_i$  etc., with the time aspect implicitly understood. Let us consider rational rates of the form  $R = m/(m+1)$  for  $m \in \mathbb{Z}^+$ . Here we divide the stream into  $T = m$  substreams of rate  $1/(m+1)$  each. Any other general rate  $R$  can be handled using  $m = \lceil R/(1-R) \rceil$ . Consider the steady state structure of  $G_1$ , i.e., after the graph has converged and when there is no more peer churn. Let there be  $n$  nodes in the system in the steady state. Then,  $G_1$  is the union of two graphs  $T_1$  and  $U_1$  described below.

**Steady state:**  $T_1$  is a directed binary tree with its root connected to the server and spans all the  $n$  nodes. It is balanced in that for every degree two node, the size of the left subtree and the right subtree differ by at most one. We call the left outgoing edge as the primary edge and the right outgoing edge as the secondary edge. The corresponding children are called primary and secondary children respectively. The degree two nodes in  $T_1$  are all close to the root of the tree, i.e., no degree one node has a directed path leading to a degree two node. Further, the chain of degree one nodes leading to the leaf, for every leaf, consists of at least  $m - 1$  nodes and at most  $2m - 2$  nodes including the leaf node. In Figure 1(a), we have illustrated  $T_1$  for  $n = 11$  and  $m = 3$ . Now, given  $T_1$ ,  $U_1$  consists of edges that connect each leaf node of  $T_1$  to the secondary child of the last degree 2 node in the path from the root to the leaf, such that, the secondary child itself does not lie in the path. For the  $T_1$  shown in Figure 1(a), the graph  $U_1$  has been illustrated in Figure 1(b). The steady state graph  $G_1$  is the union of  $T_1$  and  $U_1$  and has been shown in Figure 1(c). Finally, the nodes in  $G_1$  are labelled from the set of labels  $\{1, \dots, n\}$ . The root node connected to the server has the label 1. For any degree two node  $v$  in  $T_1$  with label  $l$  and a left subtree  $L(v)$ , its primary child has the label  $l + 1$  while the secondary child has the label  $l + |L(v)| + 1$  where  $|L(v)|$  denotes the number of nodes in  $L(v)$ . This has also been shown in Figure 1(c).

The other substream graphs  $G_2, \dots, G_m$  also have a similar topological structure, but the peers with out-degree two in each of the  $T_i$ 's are different. This is illustrated in Figure 2. Each of the  $G_i$  also has its own labeling similar to  $G_1$ , i.e., every peer has  $m$  labels associated with it for the  $m$  substreams. From

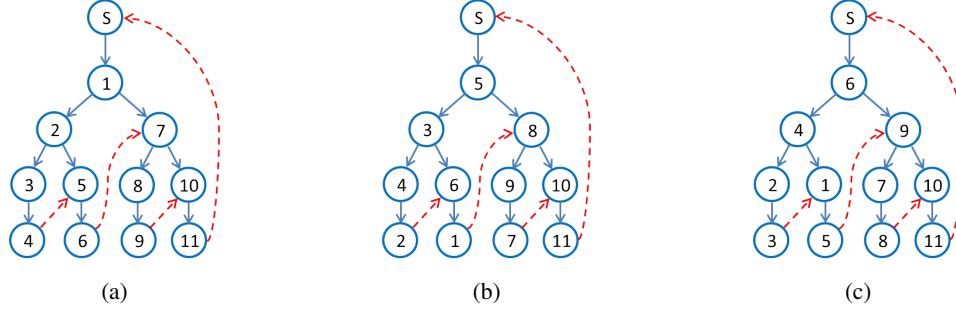


Figure 2. An example showing a steady state topology of  $G_1, G_2$  and  $G_3$  for  $n = 11$  and  $R = 3/4$  ( $m = 3$ ). The node labels shown are with respect to the first substream.

the above discussion, it is easy to see the proof of Theorem 1.

**Non steady-state:** Now, at any time instant not necessarily in the steady state,  $T_i$  as a subgraph of  $G_i$  is the shortest path graph for the peers from the server. The remaining edges form the edges in  $U_i$ . This redundancy in the form of edges in  $U_i$  is critical for the algorithm to handle peer churn. For convenience, we have summarized key characteristics of the distribution graphs by the following properties.

**Property 1.** For all  $i \in [m]$ , there exists a directed path from the server to all the peers in  $G_i$ .

**Property 2.** For all  $i \in [m]$ , (i) every node has an out-degree of either one or two in  $G_i$ , (ii) every secondary child has an incoming edge from a leaf of  $T_i$  and (iii) any node has an out-degree of two in at most one substream graph.

**Property 3.** In any steady state configuration, for all  $i \in [m]$ , we have (i) Properties 1 and 2 are satisfied, (ii) the sizes of the primary and secondary subtrees of any degree two node in  $T_i$  differ by at most one, (iii) the chain of degree one nodes in  $T_i$  have atleast  $m - 1$  nodes and at most  $2m - 2$  nodes including the leaf and (iv) no degree two peer has a degree one parent in  $T_i$ .

These properties are used in the presentation of our algorithm in section 4 and also in the subsequent sections. We now present the algorithm.

## 4. Algorithm

The algorithm consists of a set of procedures that are run in parallel (and distributedly) by each peer. Since the algorithm is flow-based, deterministic and distributed, the message forwarding (peer and piece selection) itself is straight-forward and is discussed in section 4.1. Sections 4.2 and 4.3 discuss the operations that are required for the primary goal of ensuring connectivity and bandwidth whenever peers depart or arrive. The remaining sections 4.4 and 4.5 deal with the secondary goal of balancing the topology in order to minimize the delay. For conciseness, the detailed pseudo-code and illustrative representations of the procedures are moved to Appendix A.

### 4.1. Label Control

Peer labels and addresses sent as control messages can be used by other peers to get an estimate of the sizes of their subtrees. This information is used by several procedures to follow, including the balancing subroutine. Each peer has to forward at most one (label, address) pair in each of the  $G_i$ . They cost at most  $\log_2 n_{\max}(t)$  bits per round which is minimal compared to the packet sizes. Suppose there are  $n$

nodes in the system. The node labeled  $n$  has an edge  $(n, S)$  in  $U_i$  where  $S$  is the server. Hence, node 1 knows the maximum label index  $n$  through  $S$ . Now, any degree two node in  $T_i$  sends the label of its secondary child as the control message to its primary child. It sends the label received from its parent in  $T_i$  as the control message to its primary child. A degree one node in  $G_i$  simply forwards the control message received from its parent to its child. By sending control messages as above, any node with label  $v$  in substream graph  $G_i$  receiving a control message  $l$  from its parent can know that the labels of the peers in its subtree ranges from  $v + 1$  to  $l - 1$ . For example, in Figure 2(a) node 2 receives the message  $l = 7$  since the label of the secondary child connected to node 1 is 7. As such the labels of the nodes in the subtree of node 2 range from 3 to 6.

## 4.2. Label Update

In the presence of peer churn, the labeling structure of section 3 might no longer be valid – some labels are no longer in the system, while others require new labels to be assigned to them. As such, a label update procedure constantly tries to keep the node labels updated in order to achieve the desired structure. In any substream graph the departure of a peer affects the labels of only those nodes which have a label greater than the label of the departed peer in that graph. For example in Figure 2(a) the departure of node 4 will cause the labels of nodes 5 – 11 to decrease by one. As such the labels can be updated by broadcasting the label of the departed peer and the flag “-1”, which essentially directs all nodes having a label value greater than the label of the departed peer to reduce their value by 1. This update can be performed quickly by using the edges of all  $m$  substream graphs in order to do the global broadcast.

## 4.3. Peer Churn

An important characteristic of our present structure is the ease with which peer departures can be handled. Whenever a peer with (out) degree one in  $G_i$  departs, a natural way to restore connectivity is for its child to connect to its parent. Further, the structure of the distribution graphs (Property 2) ensures that every secondary child of a degree two peer in  $G_i$ , receives an edge in  $U_i$ . As such, when a degree two peer departs, then the primary child connects to its parent, while the secondary child continues to receive the stream from the redundant edge in  $U_i$ . For example, in Figure 2(a) (i) if peer 1 departs, the edge  $(s, 2)$  is formed; 7 receives the stream from 6; (ii) if peer 5 departs, the edges  $(2, 6), (4, 6)$  are formed; (iii) if peer 6 departs, the edge  $(5, 7)$  is formed. Hence, the  $G_i$ ’s continue to satisfy Property 2 even under peer departures.

The arrival procedure, whenever a new peer enters the system, should also be such that Property 2 holds after the arrival. However, the main objective for any arriving peer is to first receive all the  $m$  substreams. Whenever a new node arrives into the system it can contact an arbitrary node. The contacted node includes the new node as its child in all  $G_i$ ’s where it has a degree one. In substreams where the new node has not yet been included (because the contacted node has a degree two), the new node can request to be the parent of one of its children from the previous substream trees. These operations preserve Properties 1 and 2. The departure and arrival procedures have been illustrated in Figures 3 and 4 respectively in Appendix A.

## 4.4. Active Balance

This procedure is used as a sub-routine in the balancing algorithm of section 4.5. Our balancing procedure is such that, even if only one of the trees  $T_i$  is balanced, it can induce its topology onto the other substream trees in a cyclic fashion. The present procedure, Active Balance, is used only when none



of the  $T_i$ 's are balanced. In this case, Active Balance tries to balance the first tree  $T_1$ , which can then balance the other trees. In this sense, it is used only as a last resort while balancing.

From section 4.1 we know that peers can estimate the size(s) of the subtree(s) below them in each of the  $T_i$ 's by using the label messages received. If the labels have been updated, for any degree two node in  $G_1$  with a label  $v$  and incoming control message  $l$  from its parent, if the label of its secondary child is not equal to  $(v + l)/2$ , then it is clear that the left and the right subtrees of  $v$  are not balanced. As such,  $v$  breaks its secondary edge and connects to the node with label  $(v + l)/2$ . Note that while  $v$  knows that it needs to connect to the node labeled  $(v + l)/2$ , it might not know the physical address in order to initiate and complete the connection. One way to do this is for  $v$  to request the physical address from the tracker server. Another alternative is to gossip the physical address of the desired node. The new links are also formed such that Property 2 holds, i.e., whenever a peer with in-degree one in  $G_i$  receives a new edge from a peer upstream, then the old edge becomes an edge in  $U_i$  while the new edge becomes the primary receiving edge in  $T_i$ . If the in-degree of the receiving peer is two, then the primary edge is broken.

#### 4.5. Induced Balance

As mentioned in the previous section 4.4, Induced Balance is the primary balancing procedure of our algorithm and includes a collection of sub-routines. Let us assume that the graph  $T_1$  is balanced. Then every leaf node in  $T_1$  has an edge in  $U_1$  going to a secondary child. We associate the degree two parent of such a secondary child with each of the degree one chain of nodes above the leaf. For example, in Figure 2(a), nodes  $\{3,4\}$ ,  $\{5,6\}$  and  $\{8,9\}$  are associated with nodes 2, 1 and 7 respectively (the last set of peers  $\{10,11\}$  are atypical and are not associated with any degree two peer). Now, the way  $G_2$  can be induced from  $G_1$  is through a series of steps in which (i) the top-most node of the chain takes the place of the degree two node, (ii) the entire chain moves up by a node and (iii) the degree two node takes the place of the leaf node. For instance, in Figure 2(b) (as induced by Figure 2(a)), node 5 has taken the place of node 1, 6 has moved up and node 1 has taken the position of the leaf node 6. Implementing these three steps at all leaf nodes ensures that the resulting  $G_2$  is structurally the same as  $G_1$  but with a fresh set of degree two nodes. This can be done in a distributed fashion, since the peers in the degree one chain receive the address of the secondary child of the associated degree two node by the label forwarding procedure (section 4.1).

As such, whenever a tree  $T_i$  is balanced, it tries to initiate the above three step procedure to induce its topology onto  $T_{i+1}$  (modulo  $m$ , i.e.,  $T_{m+1} \equiv T_1$ ). If  $T_{i+1}$  is already balanced, then such a request is turned down. Inducing the topology of  $G_{i+1}$  from a balanced  $G_i$  takes at most 5 rounds in our algorithm (Appendix A). Therefore, if at least one of the  $T_i$ 's is balanced, then in at most  $5m$  rounds, we expect all the trees to get balanced. If any degree two peer in  $T_1$  is not balanced for  $5m$  rounds, we initiate the Active Balance procedure in section 4.4 in order to balance  $T_1$ .

Also, the inducing procedure is initiated by the first node in the degree one chain such as nodes 3,5 or 8 in Figure 2(a). However, the request is made only if such nodes cannot already be a degree two node in  $G_i$  – if the degree one chain below a node is too long ( $> 2m - 2$ ), then a secondary edge is formed within  $G_i$  itself. Similarly, if either of the subtrees of a degree two node contain less than  $m - 1$  nodes, then the secondary edge is broken. This ensures that (iii) of Property 3 holds.

Thus, we have addressed the key issues of bandwidth management, connectivity and delay in the algorithm. In the following section, we give the proof of Theorem 1. The fast reconfiguration property and the linear scaling of  $K$  with  $M$  has been proved in Proposition 3 in the Appendix B.

## 5. Proof of Theorem 1

**Proposition 1.** *In any steady state configuration Property 3 is satisfied. Conversely, any  $G_1, \dots, G_m$  satisfying Property 3 are stable states for the system.*

*Proof:* Any peer that is completely disconnected from any of  $G_1, \dots, G_m$  can enter the system as a new peer by contacting the server. As such, in the steady state all peers are part of the substream graphs and satisfy Property 1. Properties 2–(i) and (iii) are locally enforced by the peers. Now, for any  $G_1, \dots, G_m$  satisfying Property 1, the forwarding of the label addresses by procedure Label Control (section 4.1) makes sure that the leaf nodes of  $T_1, \dots, T_m$  connect to their corresponding secondary children. Hence Property 2–(ii) holds. Properties 3–(ii), (iii) and (iv) follow because of the procedures in Induced Balance (section 4.5). The balancing algorithm ensures that subtrees of every degree two peer is balanced. The supplementary procedures in Induced Balance also ensure that the degree one chains are between  $m - 1$  and  $2m - 2$  nodes long as discussed in section 4.5. By the same procedure, if any degree one peer has a degree two child, then a new secondary edge is formed by the degree one peer since the subtree below it has to have larger than  $2m - 2$  peers. For the converse, consider any  $G_1, \dots, G_m$  satisfying Property 3. The only procedures that change the topology of the graphs are in Induced Balance (section 4.5). However, since the graphs are already balanced and the degree one chains have between  $m - 1$  and  $2m - 2$  nodes, neither the balancing algorithm nor the supplementary procedures change anything.  $\square$

*Proof of Theorem 1:* In the steady state, since Property 1 holds, we have that the length of the degree one chains range from  $m - 1$  to  $2m - 2$ . A balanced binary tree of depth  $d$  has  $2^{d-1}$  leaves and  $2^d - 1$  nodes. Therefore, we must have  $2^{d-1}(m - 1) + 2^d - 1 \leq n$ ,

$$\Rightarrow d \leq \log_2 \left( \frac{n + 1}{m - 1} \right) \quad (1)$$

$$\Rightarrow D \leq \log_2 \left( \frac{n + 1}{m - 1} \right) + 2m - 2 \quad (2)$$

$$= \log_2(n + 1) + \log_2(1 - R) + \frac{2R}{1 - R} - 2. \quad (3)$$

For a general upload capacity of  $C$  instead of 1, by proportionately scaling the substream rates, we have the required delay bound.  $\square$

In Section 7, we show that the above delay of the algorithm is order optimal. We now briefly discuss the scenario of a lowered redundancy in the network.

## 6. Rate-Delay-Tolerance Tradeoff

Tree based algorithms, such as [3], [10], have a delay guarantee of  $\lceil \log_2 n \rceil$  for a streaming rate of  $R = 1$ , while the algorithm we have presented has a weaker delay guarantee of order  $O(\log n + 1/(1 - R))$  (Theorem 1) for a rate  $R \leq 1$  in steady state. This can be explained by introducing a parameter called tolerance,  $\tau$ . In the streaming algorithm discussed in sections 3 – 5, we had incorporated redundant capacity into the individual substream graphs using the edges in  $U_i, i \in [m]$ . Now, consider a scenario in which the the redundancy is reduced by a factor of  $1 - \tau$  for some  $0 \leq \tau \leq 1$ , i.e., let the edges in  $U_i, \forall i \in [m]$ , have a rate of  $(1 - \tau)/m$  instead of  $1/m$  for  $R = m/(m + 1)$ . The following proposition demonstrates the gain in the delay obtained for a lowered redundancy.

**Proposition 2.** *For a tolerance parameter  $\tau$ , the steady state delay guaranteed by the algorithm is*

bounded by

$$D(R, \tau, n) \leq \log_2(n+1) - \log_2 \left( \frac{R(1-\tau)}{1-R} + 1 \right) + \frac{2R(1-\tau)}{1-R} - 2, \quad 0 \leq \tau \leq 1, \quad (4)$$

for  $n$  peers in the system and a rate of  $R$ .

*Proof:* In the steady state of the original algorithm, the peers had a degree of one in all the substream graphs or they had a degree two in one of the graphs and degree one in all the rest. By a slight modification, we can make the algorithm more symmetric where every peer with degree two in some  $T_i$  is necessarily a leaf node in some other tree in the steady state. This leads to a more even distribution of capacity, i.e., any peer has degree one in  $m-2$  trees and degree zero, two in one tree each or it has degree one in all the trees in the steady state. This corresponds to a total upload capacity of  $mr$  and  $(m-2)r + r + r(1-\tau)$  respectively, where  $r$  denotes the rate carried by each tree  $T_i$ . Therefore, we must have  $mr + (1-\tau)r \leq 1 \Rightarrow r \leq \frac{1}{m+1-\tau}$ . As such, in this scenario we can support a total rate of  $R = m/(m+1-\tau)$  across the  $m$  substream trees, which is higher than the rate  $m/(m+1)$  of our algorithm. Since the topology is the same in both cases, by substituting for  $m$  in Equation (3) for delay, we get the desired bound in Equation (4).  $\square$

Proposition 2 shows that for a rate of  $R$ , the steady state delay obtained by lowering the amount of redundancy in the system is lower. The extreme case in which there is no redundancy at all in the system, i.e.  $\tau = 1$ , corresponds to tree based algorithms with a deterministic delay of  $\lceil \log_2 n \rceil$ . Thus, we have obtained a relationship which shows the tradeoff between rate, delay and redundancy for the framework of our algorithm.

For  $\tau = 0$ , one implication of the way the substream graphs are structured (Property 2) is that connectivity of the nodes within the substream graphs (Property 1) directly translates to availability of download bandwidth from which peers can receive packets at a full rate of  $R = m/(m+1)$ . However, if we reduce the redundancy in the graphs, i.e., for  $\tau > 0$ , then with peer churn some of the peers have an upper bound of  $(1-\tau)/(m+1)$  on the substream rates, even if the graphs are connected, until the graph stabilizes. It is important to note that, there is always enough capacity for the peers in the union of the substream graphs since every peer uploads at a rate at least as much as the download rate. The substream graphs essentially introduce an asymmetry in the distribution of the capacity of each node across the different substreams in order to reduce delay. The stabilization algorithm ensures that the excess capacity available in any substream graph is effectively transferred to those in need. However, for the duration of the stabilization, even with connectivity assumptions, we can only guarantee a rate of  $(1-\tau)m/(m+1) = (1-\tau)R$  for the peers. This highlights the drawback with using a non-zero tolerance  $\tau$ ; a large tolerance parameter can cause the transient drops in the rate received to be large. Hence, the lower rate and larger delay of our algorithm, compared to the tree based algorithms mentioned in the beginning of this section, has the advantage of guaranteed continuous playback at full rate even during peer churn.

## 7. Converse

The streaming algorithm we have presented involved binary trees in the substream graphs. In general, the distribution graphs for streaming can be of any topology. However, in this section, we show that the steady state delay of our algorithm in Theorem 1, is order optimal within the general class of algorithms that use multiple arbitrarily structured graphs with redundancies for streaming.

Consider a directed tree with  $n$  nodes, where the nodes have out-degrees ranging from 0 to  $l$ . Let  $d^{(i)}$  denote the fraction of the nodes having an out-degree of  $i$ , for  $i = 0, \dots, l$ . It is clear that the tree with the lowest depth, for a given  $(d^{(0)}, \dots, d^{(l)})$ , has the largest degree nodes on the very top followed

by the second largest degree nodes and so on. A lower bound for the depth of such a tree is given by (Proposition 4 in Appendix)

$$D \geq \frac{d^{(1)}}{d^{(0)}} + \log_l \left( 1 + \sum_{k=2}^l n d^{(k)} (k-1) \right) - (l-2) \log_l \left( \frac{l!}{2} \right). \quad (5)$$

We now show that the delay in Theorem 1 is order optimal among any algorithm satisfying the conditions of Theorem 2. The full proof of Theorem 2 has been presented in Appendix B.

*Proof sketch of Theorem 2:* A general streaming algorithm can work over any connected graph of  $n$  vertices (mesh), where each vertex has an out-degree of at most  $\Delta$ . In the steady state, if communication happens via flow (copy + forward), and is deterministic, then one can always consider the flow to be an union of many constant rate sub-flows. Therefore, without loss of generality let us consider  $T$  trees with the  $i$ th tree carrying a rate of  $r_i$ . The full topology of the multicast streams can include more edges than just the trees above. The trees simply correspond to the routes by which the packets arrive earliest from the source to the peers. Now, suppose any one node departs the system; then at least one or more of the trees are broken. As such, reception of flow at full rate is hindered for some of the nodes and needs to be restored as fast as possible, if not immediately. Restoring is possible only by contacting another node in the tree corresponding to the substream, that is still connected to the server. Here, we are looking at a class of algorithms in which such a restoration is done by means of redundant links. Within this class of algorithms (that are solutions to the problem) we have the converse result stated in the theorem.

Let  $d_i^{(j)}$  denote the fraction of nodes having an out-degree of  $j$  in tree  $i$ . Clearly,

$$d_i^{(0)} + d_i^{(1)} + \dots + d_i^{(l)} = 1, \quad \forall i = 1, \dots, T. \quad (6)$$

Since any tree with  $n$  nodes has  $n-1$  edges, we have

$$n(d_i^{(1)} + 2d_i^{(2)} + \dots + (l-1)d_i^{(l-1)} + l d_i^{(l)}) = n-1, \quad \forall i = 1, \dots, T. \quad (7)$$

Now, every degree  $i$  node for  $i \geq 2$  needs atleast  $i-1$  redundant edges because of the capacity requirement in the theorem. As such, the cumulative node capacity constraint becomes

$$\sum_{i=1}^T (n-1)r_i + n(d_i^{(2)} + 2d_i^{(3)} + \dots + (l-1)d_i^{(l)})r_i \leq n. \quad (8)$$

The proof essentially obtains a lower bound for the expression in Equation (5) based on above Equations (6), (7) and (8). The delay for the  $i$ -th tree  $D_i$  can be lower bounded as

$$D_i \geq \frac{1}{d_i^{(0)}} - (\log_e(l-1) + 2) + \log_l \left( 1 + n d_i^{(0)} \right) - (l-2) \log_l \left( \frac{l!}{2} \right), \quad \forall i = 1, \dots, T. \quad (9)$$

The right hand side of the above is a decreasing function of  $d_i^{(0)}$  in  $(0, 1)$ . Equations (6), (7) and (8) also yield

$$\min_i d_i^{(0)} \leq \frac{1}{R} - 1 + \frac{2}{n} \quad (10)$$

(the proofs for Equations (9) and (10) have been discussed in Appendix B). Letting  $i^* = \arg \min d_i^{(0)}$ , the overall delay for the system can be bounded by the delay of the  $i^*$ -th tree:

$$\Rightarrow D \geq \log_l n + \frac{R}{2(1-R)} + \log_l \left( \frac{2(1-R)}{R} \right) - (l-2) \log_l \left( \frac{l!}{2} \right) - \log_e(l-1) - 2 \quad (11)$$

for  $n \geq 3R/(1-R)$ . For  $l = \Delta$  and a node capacity of  $C$  (rather than 1) replacing  $R$  by  $R/C$ , we get the desired theorem. Hence we can conclude that the steady state delay in our algorithm, Theorem (1), is order optimal for the class of algorithms satisfying the property in the theorem.  $\square$

## 8. Conclusion

We have presented a deterministic algorithm for streaming over structured distribution graphs in a peer-to-peer network. The algorithm has the peer churn handling capability of unstructured algorithms combined with the deterministic delay guarantees of structured algorithms, thus offering the best of both worlds. We have also identified a tolerance parameter, that is related to the transient rate guarantee, and have discussed its relationship to rate and delay. Continuity of streaming playback is an important quality of service metric that has been overlooked in the P2P streaming literature. For the class of algorithms we discussed, we have shown that an additional delay of  $R/(C - R)$  is the price paid for ensuring continuity. In general, other forms of adding redundancy exist – particularly coding techniques such as MDC or network coding. It would be interesting to study how these other methods interact with delay, rate and continuity. Implementing the present algorithm for practical real-world performance evaluation is also an important future direction.

## References

- [1] B. Haeupler, “Simple, fast and deterministic gossip and rumor spreading.” in *SODA*. SIAM, 2013, pp. 705–716.
- [2] V. N. Padmanabhan and K. Sripanidkulchai, “The case for cooperative networking,” in *Revised Papers from the First International Workshop on Peer-to-Peer Systems IPTPS*, 2001.
- [3] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, “Splitstream: High-bandwidth content distribution in cooperative environments,” in *Peer-to-Peer Systems II*. Springer, 2003, pp. 292–303.
- [4] L. Vu, I. Gupta, J. Liang, and K. Nahrstedt, “Mapping the p2p network: Studying the impacts of media streaming on p2p overlays,” 2006.
- [5] V. N. Padmanabhan, H. J. Wang, and P. A. Chou, “Resilient peer-to-peer streaming,” in *Network Protocols, 2003. Proceedings. 11th IEEE International Conference on*. IEEE, 2003, pp. 16–27.
- [6] D. A. Tran, K. A. Hua, and T. Do, “Zigzag: An efficient peer-to-peer scheme for media streaming,” in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 2. IEEE, 2003, pp. 1283–1292.
- [7] W. Zhang, Q. Zheng, H. Li, and F. Tian, “An overlay multicast protocol for live streaming and delay-guaranteed interactive media,” *Journal of Network and Computer Applications*, vol. 35, no. 1, pp. 20–28, 2012.
- [8] S. Liu, R. Zhang-Shen, W. Jiang, J. Rexford, and M. Chiang, “Performance bounds for peer-assisted live streaming,” in *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 1. ACM, 2008, pp. 313–324.
- [9] S. Liu, M. Chen, S. Sengupta, M. Chiang, J. Li, and P. A. Chou, “P2p streaming capacity under node degree bound,” in *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*. IEEE, 2010, pp. 587–598.
- [10] J. Zhu and B. Hajek, “Tree dynamics for peer-to-peer streaming,” *arXiv preprint arXiv:1308.1971*, 2013.
- [11] J. Kim and R. Srikant, “Real-time peer-to-peer streaming over multiple random hamiltonian cycles,” *Information Theory, IEEE Transactions on*, vol. 59, no. 9, pp. 5763–5778, 2013.

- [12] B. Doerr, T. Friedrich, and T. Sauerwald, “Quasirandom rumor spreading,” in *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '08. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2008, pp. 773–781. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1347082.1347167>
- [13] G. Giakkoupis, “Tight bounds for rumor spreading in graphs of a given conductance,” in *28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), T. Schwentick and C. Dürr, Eds., vol. 9. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2011, pp. 57–68. [Online]. Available: <http://drops.dagstuhl.de/opus/volltexte/2011/2997>
- [14] K. Censor-Hillel, B. Haeupler, J. Kelner, and P. Maymounkov, “Global computation in a poorly connected world: Fast rumor spreading with no dependence on conductance,” in *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '12. New York, NY, USA: ACM, 2012, pp. 961–970. [Online]. Available: <http://doi.acm.org/10.1145/2213977.2214064>
- [15] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup service for internet applications,” in *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '01. New York, NY, USA: ACM, 2001, pp. 149–160. [Online]. Available: <http://doi.acm.org/10.1145/383059.383071>
- [16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, *A scalable content-addressable network*. ACM, 2001, vol. 31, no. 4.
- [17] A. Rowstron and P. Druschel, “Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems,” in *Middleware 2001*. Springer, 2001, pp. 329–350.
- [18] L. Massoulie, A. Twigg, C. Gkantsidis, and P. Rodriguez, “Randomized decentralized broadcasting algorithms,” in *Proceedings of IEEE INFOCOM*, 2007.
- [19] S. Sanghavi, B. Hajek, and L. Massoulie, “Gossiping with multiple messages,” *IEEE Transactions on Information Theory*, 2007.
- [20] T. Bonald, L. Massoulie, F. Mathieu, D. Perino, and A. Twigg, “Epidemic live streaming: Optimal performance trade-offs,” in *Proceedings of ACM SIGMETRICS*, Annapolis, MD, June 2008.
- [21] N. Magharei and R. Rejaie, “Prime: Peer-to-peer receiver-driven mesh-based streaming,” *IEEE/ACM Transactions on Networking (TON)*, vol. 17, no. 4, pp. 1052–1065, 2009.
- [22] D. Kostić, A. Rodriguez, J. Albrecht, and A. Vahdat, “Bullet: High bandwidth data dissemination using an overlay mesh,” in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. ACM, 2003, pp. 282–297.
- [23] J. Munding, R. Weber, and G. Weiss, “Optimal scheduling of peer-to-peer file dissemination,” *Journal of Scheduling*, vol. 11, no. 2, pp. 105–120, 2008.
- [24] R. Kumar, Y. Liu, and K. Ross, “Stochastic fluid theory for p2p streaming systems,” in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*. IEEE, 2007, pp. 919–927.
- [25] S. Liu, M. Chen, S. Sengupta, M. Chiang, J. Li, and P. A. Chou, “P2P streaming capacity under node degree bound,” in *Proceedings of IEEE ICDCS*, 2010.
- [26] C. Law and K.-Y. Siu, “Distributed construction of random expander networks,” in *Proc. IEEE INFOCOM*, 2003.

- [27] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, “Less is more: Trading a little bandwidth for ultra-low latency in the data center,” in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI’12. Berkeley, CA, USA: USENIX Association, 2012, pp. 19–19. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2228298.2228324>
- [28] P. Babarczy, J. Tapolcai, L. Rónyai, and M. Médard, “Resilient flow decomposition of unicast connections with network coding,” *CoRR*, vol. abs/1401.6670, 2014.
- [29] J. Mundinger, R. Weber, and G. Weiss, “Analysis of peer-to-peer file dissemination amongst users of different upload capacities,” *Performance Evaluation Review, Performance 2005 Issue*, 2006.

## Appendix A.

### Algorithm

In the following we have presented the pseudo-code for the procedures discussed in sections 4.1– 4.5. Illustrations have also been included.

#### A.1.Label Control

---

**Algorithm 1** Message Forwarding algorithm for node  $v$  in  $G_i$

---

**Require:** degree of node in  $T_i$  and the addresses, labels of children;

```

1: procedure FORWARD(msg,  $l$ , add)           ▷ msg: streaming message;  $l$ : label; add: address
2:   if degree = 2 then
3:     send (msg, secondary child’s label and address) to primary child in  $T_i$ ;
4:     send (msg,  $l$ , add) to secondary child in  $T_i$ ;
5:   else if degree = 1 then
6:     send (msg,  $l$ , add) to child in  $T_i$ ;
7:   else
8:     send (msg) to child in  $U_i$ ;
9:   end if
10: end procedure

```

---

#### A.2.Label Update

---

**Algorithm 2** Label Update Algorithm for node  $v$  in all substream graphs

---

**Require:** label of parent node(s) of  $v$  in  $G_i, i = 1, \dots, m$ ;

```

1: procedure UPDATE( $i, l, f, t$ )           ▷  $l$ : label of departed/arrived node in tree  $i$  at time  $t$ ;  $f$ : flag
2:   if this label update not already done and label( $v$ )  $\geq l_i$  then           ▷ check using time-stamp  $t$ 
3:     label( $v$ )  $\leftarrow l_i + f_i$ 
4:   end if
5:   forward ( $i, l, f, t$ ) to all edges (undirected) other than the received edge in  $T_1 \cup T_2 \cup \dots \cup T_m$ ;
6: end procedure

```

---

#### A.3.Peer Churn

Figures 3 and 4 illustrate the node departure and arrival procedures respectively as discussed in section 4.3.

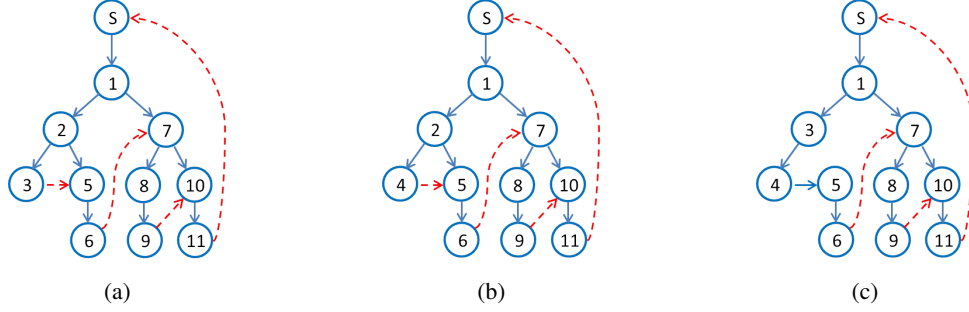


Figure 3. An example illustrating the change in the topology of  $G_1$ , from Figure 2(a), due to the departure of (a) node 4, a leaf node in  $T_1$ , (b) node 3, a degree one node in  $T_1$  and (c) node 2, a degree two node in  $T_1$ .

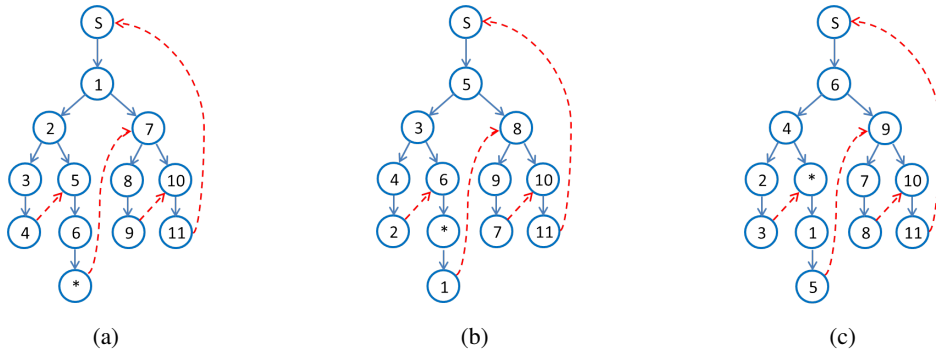


Figure 4. Topologies of  $G_1$ ,  $G_2$  and  $G_3$  respectively, due to the arrival of a new node denoted by \*. The newly arrived peer is assumed to have contacted the node 6 in Figure 2 initially.

---

**Algorithm 3** Edge update algorithm under parent departures for node  $v$  in  $G_i$

---

**Require:** labels of children in  $T_i$ ; parent(s) of  $v$  in  $T_i$  and  $U_i$  (if any) and their parents;

```

1: procedure DEPARTURE( $p$ )
2:   if  $v$  is not a secondary child of  $p$  then
3:     form the edges  $(q, v)$  and  $(r, v)$  as appropriate;  $\triangleright q, r$ :  $p$ 's parents in  $T_i, U_i$  (if any)
4:      $\text{label}(v) \leftarrow \text{label}(v) - 1$ ;
5:     broadcast label update message  $(i, \text{label}(v) - 1, -1, t)$  along all edges of  $\bigcup_{i=1}^m T_i$ ;
6:   else
7:     no action;  $(r, v)$  becomes part of  $T_i$  from  $U_i$ ;  $\triangleright r$ :  $v$ 's parent in  $U_i$ 
8:   end if
9: end procedure

```

---



---

**Algorithm 4** Edge update algorithm under node arrivals for node  $v$  in  $G_i$

---

**Require:** labels of children in  $T_i$ ; parent  $p$  of  $v$  in  $T_i$ ; parent  $r$  of  $v$  in  $U_i$  if any;

```

1: procedure ARRIVAL( $q$ ) ▷  $q$ : node that requests to become parent
2:   if  $q$  requests to insert itself between  $p$  and  $v$  then
3:     if  $\text{degree}(v) \neq 2$  in  $T_i$  then
4:       break  $(p, v)$  and form the edge  $(q, v)$  in  $T_i$ ;
5:     else if  $\text{label}(v_s) \neq \lceil (\text{label}(v) + l)/2 \rceil$  then ▷  $l$ : control label received from  $p$ 
6:       break  $(p, v)$  and form the edge  $(q, v)$  in  $T_i$ ;
7:     else reject request by  $q$  to connect to  $v$ ; retain old edge  $(p, v)$ ;
8:     end if
9:   else if  $q$  requests to insert itself between  $r$  and  $v$  then
10:    break  $(r, v)$  and form the edge  $(q, v)$  in  $U_i$ ;
11:    give  $p$ 's address to  $q$ ; ▷  $q$  then requests to insert itself above  $p$  in tree  $T_{i+1}$ 
12:  end if
13:  if  $\text{label}(v)$  not consistent with label of parent(s) then update label and broadcast the label update
    message; ▷ see procedure DEPARTURE
14: end procedure

```

---

#### A.4.Active Balance

---

**Algorithm 5** Balancing of a degree two node  $v$  in tree  $T_1$

---

**Require:** labels of primary child  $v_p$  and secondary child  $v_s$  in  $T_1$ ; parent  $p$  of  $v$  in  $T_1$ ;

```

1: procedure ACTIVEBALANCE( $l$ ) ▷  $l$ : control label received from parent in  $T_1$ 
2:   if  $\text{label}(v_s) \neq \lceil (\text{label}(v) + l)/2 \rceil$  and  $p$  is balanced then
3:     break the edge  $(v, v_s)$  in  $T_1$ ;
4:     find the node  $u$  have the label  $\lceil (\text{label}(v) + l)/2 \rceil$  in  $G_i$ ;
5:     form the edge  $(v, u)$ ;
6:   end if
7: end procedure

```

---

#### A.5.Induced Balance

In section 4.5 we have discussed a three step procedure by which peers in a balanced graph  $G_i$  can induce its topology to a subsequent unbalanced  $G_{i+1}$ . The procedures REQUEST, RESPOND and INDUCE are used to implement this.

By the control label forwarding algorithm FORWARD, in algorithm 1, all degree one children of degree two parents receive the addresses of the secondary children of their corresponding degree two nodes in  $G_i$  (such as node 3 receiving the address of node 5 in Figure 2(a)). As such, in the first step, these nodes contact those secondary children. This is presented in Algorithm 6 as the procedure REQUEST. Since all secondary children of degree two nodes receive such a request, they can exchange this information among their neighbors in order to let the requesting degree one nodes know who their prospective parents will be in the subsequent graph. Indeed, procedure RESPOND in Algorithm 7 returns the address of the prospective parent and position (primary or secondary) in the subsequent tree to any requesting degree one node. In the last step, the degree one nodes request to insert themselves in the edge between the prospective parent and child in the subsequent tree (returned by RESPOND). This is done by procedure INDUCE in Algorithm 8. Procedure REQUEST takes 1 time slot, while procedures RESPOND and INDUCE takes 3 and 1 time slots respectively. As such, the entire operation occupies at most 5 time slots.

**Algorithm 6** Step 1 of algorithm for node  $v$  to form the induced graph edges in  $G_{i+1}$  from  $G_i$

```

1: procedure REQUEST( $q$ )
2:   if  $\text{degree}(v) = 1$  and  $\text{degree}(p) = 2$  in  $T_i$  then
3:     if label difference is  $> 2m - 2$  in  $T_i$  then
4:       create secondary edge
5:     else if  $p$  is balanced then
6:       make an induced graph request to  $q$ ;
7:     end if
8:   else if  $\text{degree}(v) = 2$  and  $(l - \text{label}(v_s) < m - 1$  or  $\text{label}(v_s) - \text{label}(v_p) < m - 1)$  then
9:     break the edge to  $v_s$ ;
10:  end if
11: end procedure

```

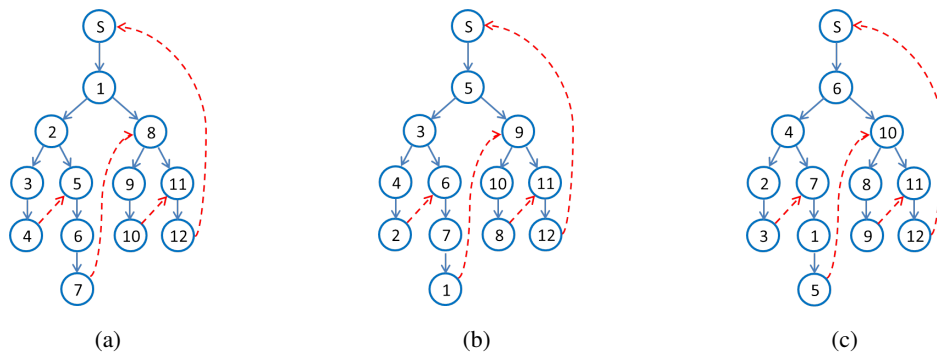


Figure 5. A possible steady state topology for  $G_1$ ,  $G_2$  and  $G_3$  respectively for  $n = 12$ . Notice that Property 3 holds.

Note that INDUCE makes an insertion request to the child of its prospective parent in  $G_{i+1}$ . The ARRIVAL routine running in the prospective child allows the insertion to take place only if the prospective child is unbalanced (if its a degree two node). This step ensures that if  $G_{i+1}$  is already well balanced, then it is not perturbed by  $G_i$ . This has been illustrated in Figure 5, where node 7 in  $G_3$  tries to break the edge  $(S, 1)$  in  $G_1$  but is refused. Similarly, the procedure REQUEST makes a request only if it cannot already be a degree two node in  $G_i$ . For example, if the degree one chain below a node is too long ( $> 2m - 2$ ), then it can form a secondary edge within  $G_i$  itself. In addition, a request is made only if the parent of the requesting node is balanced, to ensure that if  $G_i$  is ill balanced then it does not propagate its structure to  $G_{i+1}$ . REQUEST also makes sure that the degree one chains are at least  $m - 1$  nodes long, by breaking secondary edges if either of the subtrees contain less than  $m - 1$  nodes.

---

**Algorithm 7** Algorithm for node  $v$  to respond to an induced graph request

---

**Require:** parent  $p$  and children of  $v$  in  $T_i$ ;

```
1: procedure RESPOND( $q$ ) ▷  $q$  makes the request to  $v$ 
2:   send  $q$ 's address to  $p$ ;
3:   if degree( $v$ ) = 2 then send address received from  $v_s$  to  $v_p$ ;
4:   end if
5:   if degree( $v$ ) = 2 and  $v$  is a primary child then
6:     flag  $\leftarrow$  primary;
7:     send (address received from  $p$ , flag) to  $v_s$ ;
8:   else if degree( $v$ ) = 2 and  $v$  is a secondary child then
9:     flag  $\leftarrow$  secondary;
10:    send ( $q$ 's address, flag) to  $v_s$ ;
11:   else no action;
12:   end if
13:   return (address, flag) received from  $p$  to  $q$ ;
14: end procedure
```

---

---

**Algorithm 8** Algorithm for node  $v$  to form the induced graph edges in  $G_{i+1}$  from  $G_i$ 

---

**Require:** return values (address  $q$ , flag) of procedure REQUEST

```
1: procedure INDUCE( $q$ , flag)
2:   request the flag child of  $q$  to insert itself between  $q$  and flag child of  $q$ ;
3: end procedure
```

---

is balanced (REQUEST) makes sure that an ill balanced graph cannot induce its structure onto a well balanced subsequent substream graph. Noting that it can take up to  $m$  rounds for the cyclic inducing process to propagate from one graph to all the remaining, we have the following balancing algorithm.

---

**Algorithm 9** Balancing algorithm for node  $v$  in  $G_1$ 

---

**Require:** labels of primary child  $v_p$  and secondary child  $v_s$  in  $T_1$ ;

```
1: procedure INDUCEDBALANCE( $l$ ) ▷  $l$ : control label received from parent in  $T_1$ 
2:   if label( $v_s$ )  $\neq \lceil (\text{label}(v) + l)/2 \rceil$  for  $T_{\text{count}} \geq T_{\text{threshold}}$  then ▷ with  $T_{\text{threshold}} = 5m$ 
3:     run ACTIVEBALANCE( $l$ );
4:   else if label( $v_s$ )  $\neq \lceil (\text{label}(v) + l)/2 \rceil$  for  $T_{\text{count}} < T_{\text{threshold}}$  then
5:      $T_{\text{count}} \leftarrow T_{\text{count}} + 1$  in the next time slot;
6:   else  $T_{\text{count}} \leftarrow 0$ ;
7:   end if
8: end procedure
```

---

That is, we wait  $T_{\text{threshold}} = 5m$  rounds before breaking any secondary edge to form a new secondary edge. If after  $5m$  time slots the labels are still incorrect, then the node breaks its secondary edge in tree  $T_1$  as per ACTIVEBALANCE. This is because, if atleast one of the trees is balanced initially, then that tree can initiate the rearrangement cycle across all substream trees which takes at most  $5m$  rounds. If none of the trees are balanced initially, then by initiating ACTIVEBALANCE tree  $T_1$  gets balanced, which in turn causes the other trees to get balanced.

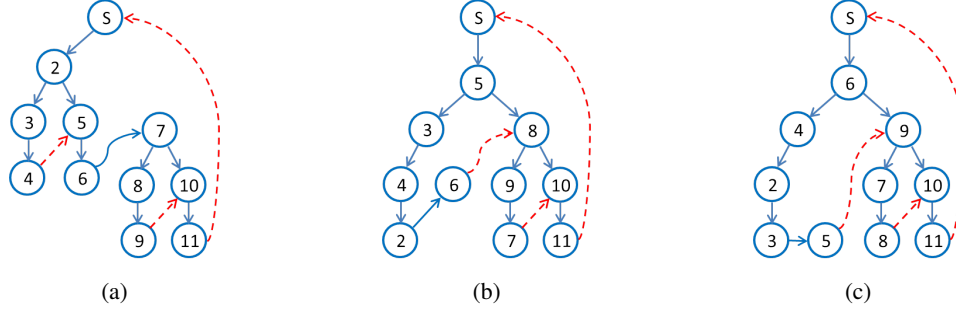


Figure 6. Topology of (a)  $G_1$ , (b)  $G_2$  and (c)  $G_3$  resulting from the departure of node 1 from Figure 2. Notice that the edges  $(3, 6)$  in  $G_2$  and  $(4, 5)$  in  $G_3$  have been broken to preserve the minimum length of 2 for the degree one chains.

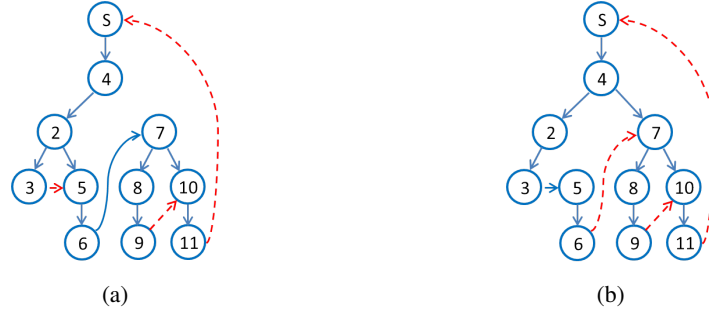


Figure 7. The process of node 4 in  $G_3$  of Figure 6 becoming a degree two node in  $G_1$ .

## A.6. Multiple Departures

So far we have been considering peer departures where only a single peer leaves the system at a time. Since we have not imposed any departure restrictions in our model, in general peers could depart in an arbitrary fashion including adversarial. Under such a scenario, the primary objective for the peers is to ensure connectivity in all the substream graphs. For example, in the network shown in Figure 2 if the nodes 1, 3, 4, 5 and 6 all leave at the same time then the node 2 is completely disconnected in all the graphs. As such, the performance is dictated by the amount of stored physical addresses,  $M$ , of the peers currently in the system. If none of the peers in the memory are available, then the node has to contact the server and re-enter the system as a new peer.

## Appendix B. Proofs

Consider the set of peers departing at time  $t$ . In the following proposition we show that the algorithm has the required resilience capability under churn. We remind that  $M$  is the total amount of memory available in each peer, while  $m$  is the number of substream graphs for a rate  $R = m/(m+1)$ .

**Proposition 3.** *If the peer departure blocks in each round are of size at most  $K = 1$ , then for a memory of  $M = m$ , the substream graphs  $G_1, \dots, G_m$  always satisfy Properties 1 and 2. In general, the algorithm requires a memory of  $M = Km$ .*

*Proof:* Note that the sizes of the sets of departing peers that are connected in  $G_i$  is also bounded

by  $K$  for any  $i$ . In section 4.3 we have discussed the case of a single peer departure. Since the repairing procedure upon departure of any peer involves only the parent and children of the peer, arbitrary departures with bounded (by 1) departure block sizes can also be handled similarly. Now, if  $G_1, \dots, G_m$  satisfy Properties 1 and 2, then it is easy to see that the repaired graphs resulting from a departure event also satisfy Properties 1 and 2. Hence by induction we get the desired result for  $K = 1$ . In general, connectivity of the peers is ensured if they have sufficient amount of memory to form new connections. For a memory of  $M = Km$ , every peer can know the address of the  $K$  parent nodes above it in each  $G_i$ . Here we ignore the secondary edges and consider the  $K$  parents in the resulting cycle graph (such as  $1-2-3-4-\dots-11$  in Figure 2(a)). Now, if the peer departure blocks consist of at most  $K$  nodes, then in the worst case a block consists of the  $K$  parents of a peer. But in this case the peer can immediately restore connectivity by making a connection to the  $K$ -th parent above it. Hence, the proposition follows.  $\square$

Since the distribution graphs  $G_1, \dots, G_m$  always satisfy Property 1, and their edges can support a rate of  $1/(m+1)$  we conclude that peers suffer a loss of packets in at most one time slot required for the reconfiguration.

In the following, we present the proof of the lower bound for the tree depth mentioned in Equation (5).

**Proposition 4.** *Any directed tree with  $n$  nodes, and where  $d^{(i)}$  fraction of the nodes have an out-degree of  $i$  for  $i = 0, 1, \dots, l$ , has a depth  $D$  that is bounded as*

$$D \geq \frac{d^{(1)}}{d^{(0)}} + \log_l \left( 1 + \sum_{k=2}^l n d^{(k)} (k-1) \right) - (l-2) \log_l \left( \frac{l!}{2} \right). \quad (12)$$

*Proof:* It is clear that the tree with the lowest depth, for a given  $(d^{(0)}, \dots, d^{(l)})$ , has the largest degree nodes on the very top followed by the second largest degree nodes and so on. Let us call a layer of nodes at a particular depth as an  $i$ -layer if the largest degree node present in that layer has the degree  $i$ . Further, let  $d_i, i = 1, \dots, l$  denote the number of the  $i$ -layers in the tree. Therefore,

$$D = \sum_{i=0}^l d_i \quad (13)$$

gives the depth of the tree. The proof proceeds by bounding the depth of each layer. The number of nodes in the topmost layer of the graph, layer  $l$ , can be bounded as

$$1 + l + \dots + l^{d_l-2} \leq n d^{(l)} \leq 1 + l + \dots + l^{d_l-1} \quad (14)$$

(If no such  $d_l$  exists, then  $d_l = 0$ ). This yields

$$\log_l(n d^{(l)}(l-1) + 1) \leq d_l, \quad (15)$$

$$l^{d_l} \leq (n d^{(l)}(l-1) + 1)l. \quad (16)$$

Now, in the second layer where there are nodes of degree  $l-1$  (or possibly lesser), since  $l^{d_l}$  constitutes an upper bound on the number of degree  $l$  parents of degree  $l-1$  nodes and  $l^{d_l-1}(l-1)$  constitutes a lower bound, we must have

$$l^{d_l-1}(l-1)(1 + (l-1) + \dots + (l-1)^{d_{l-1}-2}) \leq n d^{(l-1)} \leq l^{d_l}(1 + (l-1) + \dots + (l-1)^{d_{l-1}-1}) \quad (17)$$

$$\Rightarrow l^{d_l-1}(1 + (l-1) + \dots + (l-1)^{d_{l-1}-2}) \leq n d^{(l-1)} \leq l^{d_l}(1 + (l-1) + \dots + (l-1)^{d_{l-1}-1}). \quad (18)$$

This yields

$$\log_{l-1} \left( \frac{n d^{(l-1)}(l-2)}{l^{d_l}} + 1 \right) \leq d_{l-1}, \quad (19)$$

$$l^{d_l}(l-1)^{d_{l-1}} \leq (n d^{(l-1)}(l-2) + n d^{(l)}(l-1) + 1)(l)(l-1). \quad (20)$$

Using Equation (16) in (19) we have,

$$\log_{l-1} \left( \frac{nd^{(l-1)}(l-2)}{(nd^{(l)}(l-1)+1)l} + 1 \right) \leq d_{l-1}. \quad (21)$$

Similarly, we have in the  $(l-2)$ th layer,

$$l^{d_{l-1}}(l-1)(l-1)^{d_{l-1}-1}(l-2)(1+(l-2)+\dots+(l-2)^{d_{l-2}-2}) \leq nd^{(l-2)} \quad (22)$$

$$\Rightarrow l^{d_{l-1}}(l-1)^{d_{l-1}-1}(1+(l-2)+\dots+(l-2)^{d_{l-2}-2}) \leq nd^{(l-2)} \quad (23)$$

$$\text{and } nd^{(l-2)} \leq l^{d_l}(l-1)^{d_{l-1}}(1+(l-2)+\dots+(l-2)^{d_{l-2}-1}), \quad (24)$$

yielding

$$\log_{l-2} \left( \frac{nd^{(l-2)}(l-3)}{l^{d_l}(l-1)^{d_{l-1}}} + 1 \right) \leq d_{l-2} \quad (25)$$

$$l^{d_l}(l-1)^{d_{l-1}}(l-2)^{d_{l-2}} \leq (nd^{(l-2)}(l-3) + nd^{(l-1)}(l-2) + nd^{(l)}(l-1)+1)(l)(l-1)(l-2). \quad (26)$$

Using Equation (20) we have:

$$\log_{l-2} \left( \frac{nd^{(l-2)}(l-3)}{(nd^{(l-1)}(l-2) + nd^{(l)}(l-1)+1)(l)(l-1)} + 1 \right) \leq d_{l-2}. \quad (27)$$

We continue this process for all the  $i$ -layers for  $i \geq 2$ . Finally, in the last layer the number of degree one chains is equal to the number of the leaves. As such, we must have

$$\frac{nd^{(1)}}{nd^{(0)}} - 1 \leq d_1, \quad (28)$$

$$d_0 = 1. \quad (29)$$

Therefore, from Equation (13) we have depth

$$D \geq \frac{d^{(1)}}{d^{(0)}} + \sum_{k=2}^l \log_k \left( 1 + \frac{nd^{(k)}(k-1)}{(1 + \sum_{k'=k+1}^l nd^{(k')}(k'-1)) \prod_{k''=k+1}^l (k'')} \right). \quad (30)$$

Now, the second term in the right-hand side of Equation (30), denoted by  $T$ , can be lower bounded as

$$T \geq \sum_{k=2}^l \log_l \left( 1 + \frac{nd^{(k)}(k-1)}{(1 + \sum_{k'=k+1}^l nd^{(k')}(k'-1)) \prod_{k''=k+1}^l (k'')} \right) \quad (31)$$

$$= \log_l \prod_{k=2}^l \left( 1 + \frac{nd^{(k)}(k-1)}{(1 + \sum_{k'=k+1}^l nd^{(k')}(k'-1)) \prod_{k''=k+1}^l (k'')} \right) \quad (32)$$

$$= \log_l \prod_{k=2}^l \left( \frac{(1 + \sum_{k'=k+1}^l nd^{(k')}(k'-1)) \prod_{k''=k+1}^l (k'') + nd^{(k)}(k-1)}{(1 + \sum_{k'=k+1}^l nd^{(k')}(k'-1)) \prod_{k''=k+1}^l (k'')} \right) \quad (33)$$

$$\geq \log_l \prod_{k=2}^l \left( \frac{(1 + \sum_{k'=k+1}^l nd^{(k')}(k'-1)) + nd^{(k)}(k-1)}{(1 + \sum_{k'=k+1}^l nd^{(k')}(k'-1)) \prod_{k''=k+1}^l (k'')} \right) \quad (34)$$

$$= \log_l \left( \left( 1 + \sum_{k'=2}^l nd^{(k')}(k'-1) \right) \prod_{k=2}^l \left( \frac{1}{\prod_{k''=k+1}^l (k'')} \right) \right) \quad (35)$$

$$\geq \log_l \left( 1 + \sum_{k'=2}^l nd^{(k')}(k'-1) \right) - (l-2) \log_l \left( \frac{l!}{2} \right) \quad (36)$$

thus proving the claim.  $\square$

We now present the proofs of Equations (9) and (10) from section 7. For the sake of completeness we have presented the full-proof of Theorem 2.

*Proof of Theorem 2:* Without loss of generality let us consider  $T$  trees with the  $i$ th tree carrying a rate of  $r_i$ . This is justified because if the flow is granular we can associate a shortest path tree with each of the substreams. The full topology of the multicast streams itself can be bigger than the trees above. The trees simply correspond to the routes by which the packets arrive earliest from the source to the peers. Let  $d_i^{(j)}$  denote the fraction of nodes having an out-degree of  $j$  in tree  $i$ . Clearly,

$$d_i^{(0)} + d_i^{(1)} + \dots + d_i^{(l)} = 1, \quad \forall i = 1, \dots, T. \quad (37)$$

Since any tree with  $n$  nodes has  $n - 1$  edges, we have

$$\begin{aligned} n(d_i^{(1)} + 2d_i^{(2)} + \dots + (l-1)d_i^{(l-1)} + ld_i^{(l)}) &= n-1, \quad \forall i = 1, \dots, T, \\ \Rightarrow d_i^{(1)} + 2d_i^{(2)} + \dots + (l-1)d_i^{(l-1)} + ld_i^{(l)} &= 1 - \frac{1}{n}, \quad \forall i = 1, \dots, T. \end{aligned} \quad (38)$$

Now, every degree  $i$  node for  $i \geq 2$  needs atleast  $i-1$  redundant edges because of the capacity requirement of the theorem. As such, the cumulative node capacity constraint becomes

$$\begin{aligned} \sum_{i=1}^T (n-1)r_i + n(d_i^{(2)} + 2d_i^{(3)} + \dots + (l-1)d_i^{(l)})r_i &\leq n \\ \Rightarrow \sum_{i=1}^T \left(1 - \frac{1}{n} + d_i^{(2)} + 2d_i^{(3)} + \dots + (l-1)d_i^{(l)}\right) r_i &\leq 1. \end{aligned} \quad (39)$$

The proof essentially obtains a lower bound for the expression in Equation (5) based on above Equations (37), (38) and (39). Subtracting Equation (37) from (38) gives

$$d_i^{(2)} + 2d_i^{(3)} + \dots + (l-1)d_i^{(l)} = d_i^{(0)} - \frac{1}{n}, \quad \forall i = 1, \dots, T. \quad (40)$$

From the above, we have

$$d_i^{(j)} \leq \frac{1}{j-1} \left( d_i^{(0)} - \frac{1}{n} \right), \quad (41)$$

and combined with Equation (37) we get

$$\begin{aligned} 1 &\leq d_i^{(0)} + d_i^{(1)} + \left( d_i^{(0)} - \frac{1}{n} \right) \left( 1 + \frac{1}{2} + \dots + \frac{1}{l-1} \right) \\ &\leq d_i^{(0)} + d_i^{(1)} + \left( d_i^{(0)} - \frac{1}{n} \right) (\log_e(l-1) + 1) \\ \Rightarrow d_i^{(1)} &\geq 1 - d_i^{(0)} (\log_e(l-1) + 2) + \frac{1}{n} (\log_e(l-1) + 1) \\ \Rightarrow \frac{d_i^{(1)}}{d_i^{(0)}} &\geq \frac{1}{d_i^{(0)}} - (\log_e(l-1) + 2) + \frac{1}{nd_i^{(0)}} (\log_e(l-1) + 1). \end{aligned} \quad (42)$$

Also, the second term in the delay lower bound in Equation (5) becomes

$$\log_l \left( 1 + \sum_{k=2}^l nd^{(k)}(k-1) \right) = \log_l \left( 1 + nd_i^{(0)} \right). \quad (43)$$

As such, using Equations (42), (43) and (5) the delay for the  $i$ -th tree  $D_i$  can now be lower bounded as

$$D_i \geq \frac{1}{d_i^{(0)}} - (\log_e(l-1) + 2) + \log_l \left( 1 + nd_i^{(0)} \right) - (l-2) \log_l \left( \frac{l!}{2} \right), \quad \forall i = 1, \dots, T. \quad (44)$$

The derivative of the right-hand side above in Equation (44) with respect to  $d_i^{(0)}$  is given by

$$-\frac{1}{(d_i^{(0)})^2} + \frac{n}{(1 + nd_i^{(0)}) \log l} \quad (45)$$

which is strictly negative in  $0 < d_i^{(0)} < 1$ . As such, the minima in the right-hand side of Equation (44) is achieved by the largest achievable  $d_i^{(0)}$ . Now, using Equations (37) and (38) in (39) we get

$$\begin{aligned} \sum_{i=1}^T \left( 1 - \frac{2}{n} + d_i^{(0)} \right) r_i &\leq 1, \\ \Rightarrow \min_i d_i^{(0)} &\leq \frac{1}{R} - 1 + \frac{2}{n}. \end{aligned} \quad (46)$$

Letting  $i^* = \arg \min d_i^{(0)}$ , the overall delay for the system can be bounded by the delay of the  $i^*$ -th tree. Hence, substituting Equation (46) in (44) we have

$$\begin{aligned} D &\geq \frac{1}{d_{i^*}^{(0)}} - (\log_e(l-1) + 2) + \log_l \left( 1 + nd_{i^*}^{(0)} \right) - (l-2) \log_l \left( \frac{l!}{2} \right) \\ &\geq \frac{1}{\frac{1}{R} - 1 + \frac{2}{n}} + \log_l \left( 1 + n \left( \frac{1}{R} - 1 + \frac{2}{n} \right) \right) - (l-2) \log_l \left( \frac{l!}{2} \right) - \log_e(l-1) - 2 \\ &\geq \log_l n + \frac{R}{2(1-R)} + \log_l \left( \frac{2(1-R)}{R} \right) - (l-2) \log_l \left( \frac{l!}{2} \right) - \log_e(l-1) - 2 \end{aligned} \quad (47)$$

for  $n \geq 3R/(1-R)$ . For  $l = \Delta$  and a node capacity of  $C$  (rather than 1) replacing  $R$  by  $R/C$ , we get the desired theorem. Hence we can conclude that the steady state delay in our algorithm, Theorem (1), is order optimal for the class of algorithms satisfying the conditions of Theorem 2.  $\square$

## Appendix C.

### All-Cast

In the all-cast scenario, each peer in the system can have an independent data stream for broadcasting to all the other peers. The symmetry of the distribution topology that we constructed for the broadcast problem in sections 3 – 5 allows us to reuse the topology for all-cast. Let us assume the node capacities of the peers proportionally scale as the number of streaming sources in the system. For example, if there are  $k$  independent broadcasts then we will assume that the peers can support a total upload rate of  $k$ . The rate of each independent stream and its substreams are the same as in the original algorithm. In the single source broadcast graph the edges carrying the data streams were directed. However, since the edges of the P2P network have been assumed to be undirected in our model in section 2, we allow data transfer to happen both directions between any of the neighbours in the substream graphs  $G_i, i = 1, \dots, m$ . As such, let  $\bar{G}_i$  denote the undirected version of the directed graph  $G_i$  for all  $i$ . Then, for any source node  $v$ , the problem is to find a rooted (at the source), low depth, directed spanning tree (where the edges point away from the source) in  $\bar{G}_i$  subject to the node capacity constraints on the peers. One way to ensure the capacity constraints is to find a route in  $\bar{G}_i$ , for each independent broadcast stream, such that the out-degree for the peers is the same as in  $G_i$  for all  $i$ .



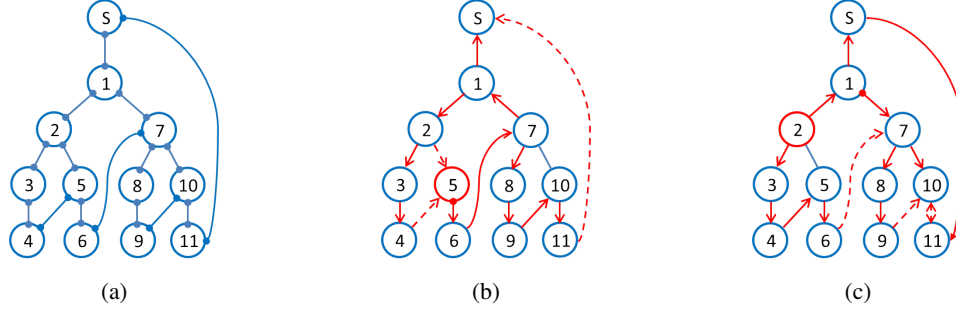


Figure 8. (a) The undirected network corresponding to the directed graph in Figure 2(a), (b) route taken by the stream if node 5 is the source and (c) route taken by the stream for node 2 as the source.

This can be done as follows. Consider the single source broadcast algorithm for a rate  $R = m/(m+1)$ . This results in the construction of  $m$  substream graphs  $G_1, \dots, G_m$ . Let  $v$  be any peer sourcing a data stream. For substream  $i$ , if  $v$  is a degree two node in  $T_i$ , then  $v$  sends the substream to its primary child and parent in  $T_i$ . Otherwise, if  $v$  is of degree one or zero, it sends the stream to its child in  $T_i$  or  $U_i$  respectively. Now, for any node  $u$  that is receiving a substream from its neighbor, if  $u$  receives it from any of the neighbors in  $T_i$ , it forwards the substream to the other neighbors in  $T_i$ . If  $u$  is a leaf-node in  $T_i$  receiving messages from its parent, then  $u$  forwards the messages to its child in  $U_i$ . On the other hand, if  $u$  receives the substream from its neighbor in  $U_i$ , then if  $u$  is of degree two in  $T_i$  and  $u$ 's parent in  $T_i$  has not yet received the stream then it forwards it to its parent and primary child in  $T_i$ . If  $u$ 's parent has received the stream, then it forwards to its children in  $T_i$ . If  $u$  has degree one, it forwards the message to its child in  $T_i$ . This algorithm has been presented in Algorithm 10 and illustrated in Figure 8. In all of the above operations, the amount of upload done by the nodes for each substream of each independent stream is the same as in the original algorithm.

---

**Algorithm 10** Algorithm for node  $v$  to source / forward an all-cast message in  $G_i$

---

**Require:** all neighbors in  $G_i$ ;

**procedure** ALLCAST

Case:  $v$  is the source of msg

**if** degree( $v$ ) = 2 **then**

    send msg to primary child and parent in  $T_i$ ;

**else**

    send msg to the child in  $T_i$  or  $U_i$ ;

**end if**

Case:  $v$  receives msg from a neighbor

**if** degree( $v$ ) = 2 **then**

    if msg was received from a neighbor in  $T_i$ , forward msg to the other two neighbors in  $T_i$ ;

    if msg was received from a neighbor in  $U_i$  and the parent in  $T_i$  has not yet received msg, then forward msg to primary child and the parent in  $T_i$ ; else forward msg to primary and secondary children in  $T_i$ ;

**else**

    forward msg to the other neighbor in  $T_i$  or  $U_i$ ;

**end if**

**end procedure**

---

**Proposition 5.** *For a substream graph  $G_i, i \in [m]$  satisfying Properties 1 and 2, procedure ALLCAST ensures that all the nodes in  $G_i$  receive the substream for any source of the broadcast.*

*Proof:* Suppose a node or a set of nodes do not receive the stream. Then we can always find a node in that set whose parent in  $T_i$  or  $U_i$  have received the stream. It cannot happen that any parent in  $U_i$  received the stream and the child did not. As such, the only possibility is that the node is a secondary child of a degree two node in  $T_i$ . But in this case, the primary child of the parent has received the stream. Now, since the node is a secondary child it also has a parent in  $U_i$  which has not received the stream (otherwise the node would have received it). There exists a directed path comprising of only primary edges, and tolerance edges from the primary child to the secondary child. Going backwards along this path implies the primary child did not get the stream, which is a contradiction.  $\square$

**Proposition 6.** *In the steady state with  $n$  nodes, the delay of any of the streams in the all-cast is bounded by*

$$D_{all-cast} \leq 2 \log_2(n+1) + \frac{8R}{1-R} + 2 \log_2(1-R) - 8. \quad (48)$$

*Proof:* For any degree two node in  $T_i$ , let  $\bar{T}_i$  and  $\underline{T}_i$  denote the tree above and below the node respectively. From Equation (1), the depth of the degree two portion of  $T_i$  in the steady state is bounded by  $d \leq \log_2 \left( \frac{n+1}{m+1} \right)$  and by Property 3, the length of the degree one chains are at most  $2m-2$ . Now, it takes at most  $2d+2m-2$  delay for the stream to reach all the nodes in  $\bar{T}_i$ . For  $\underline{T}_i$  it takes at most  $2d+2(2m-2)$  delay. Therefore, it takes at most  $2d+2(2m-2)$  delay for any degree node that is a source. Now, if any degree one node is the source, then it takes at most  $2(2m-2)$  rounds to reach a degree two node. From there on it behaves as if the degree two node is the source and hence takes at most  $2d+2(2m-2)$  delay. Hence, the net delay bounded by  $2d+4(2m-2) \leq 2 \log_2(n+1) + \frac{8R}{1-R} + 2 \log_2(1-R) - 8$  as required.  $\square$

## Appendix D. Heterogeneous Capacities

In a setting where peers have heterogeneous upload capacities, it is easily seen that the maximum possible streaming capacity is equal to the sum upload capacity of the peers divided by the number of peers [29]. Likewise, a (weak) lower bound for the maximum delay is  $\Omega(\log n)$  under constant node degree bounds. Intuitively it seems possible to be able to trade one quantity for the other, such as rate for delay etc. However, precisely characterizing the rate-delay-continuity tradeoff (analogous to section 6) in the heterogeneous case remains an important future direction.

In this section we contribute to the above question, by considering the “low-rate low-delay” regime (at zero-tolerance,  $\tau = 0$ ). Without loss of generality, let the peers have an upload capacity greater than or equal to 1. Then this regime corresponds to streaming at a rate of  $R \leq 1$ . The other direction is the “high-rate high-delay” regime, and corresponds to transmission at an optimal (or near-optimal) rate as discussed above. We have not considered this direction, and leave it for future work. The low rate regime is similar in spirit to [10], where a few dedicated high capacity peers (or servers) assist in faster data dissemination by being located in the top of the distribution trees.

The key idea here is to cluster together nodes of similar upload capacities and run the original algorithm on the clusters separately. Let us first consider the homogeneous case, as before, but with multiple source nodes providing the data stream instead of just one in each of the  $G_i$ ’s.

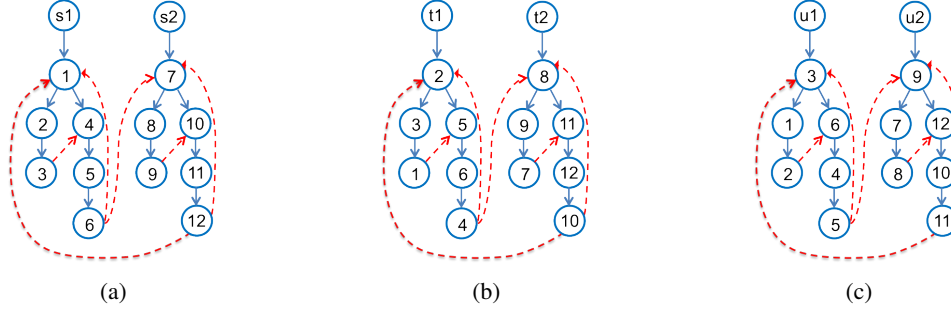


Figure 9. Example of a cluster  $C_1$  comprising of 12 nodes with 2 source nodes. (a), (b) and (c) show the three substream graphs for  $R = 3/4$ .

### D.1. Multiple Source Nodes

In our algorithm for the streaming model of section 2, for a rate of  $R = m/(m+1)$ , the server provided the stream to a single node in the substream graphs  $G_1, \dots, G_m$ . The receiving nodes have a label 1 in their respective substream labeling in the steady state (Figure 2). Now, let us suppose there are  $k_1 \geq 1$  servers providing the substream for  $G_1$ . In this case, we can expect the topology to comprise of  $k_1$  balanced graphs (satisfying Properties 1 and 2) in the steady state. The steady state topology of  $G_1, G_2$  and  $G_3$  with 12 peers and 2 sources has been illustrated in Figure 9. We have remarked that the end nodes of the substream graphs (such as nodes 10, 11 in Figure 2) are atypical and do not use their full upload capacity across  $G_1, \dots, G_m$ . However, while dealing with multiple servers, the extra capacity in the end nodes can be used to connect to both the root node the parent tree and the subsequent tree. The label information forwarded along these edges can be used for balancing the parent tree and also for ensuring that the trees are of similar size.

### D.2. Streaming in Clusters

Let  $C_1$  and  $C_{1+}$  denote the set of peers with upload capacity 1 and strictly larger than 1 respectively. We assume that whenever new peers arrive they can obtain the address of an arbitrary peer in their respective clusters. Then, for a rate of  $R = m/(m+1)$ , we let the peers in  $C_{1+}$  form and maintain the distribution graphs exactly as before in section 4. However, since the peers have an upload capacity strictly larger than 1, this does not use all of their capacity. The remaining capacity available in those nodes are used as sources for the peers in the cluster  $C_1$  as in the previous section D.1. One way to do this is to let the degree one children of degree two nodes use all of their extra capacity for sourcing that substream to  $C_1$ . In Figure 10 we have illustrated this for a cluster  $C_{1+}$  where every peer has an upload capacity of  $5/4$  for a rate  $R = 3/4$ . Since each substream is of rate  $1/4$ , the peers in  $C_{1+}$  can support up to 5 outgoing edges. While 4 edges are used for the construction of the substream graphs, the remaining edges (shown by dotted lines in the Figure) are used as source nodes for the lower capacity cluster  $C_1$ . For example, if  $C_1$  is as in Figure 9, then peers 2, 4 in Figure 10 can be the sources  $s_1, s_2$  in Figure 9 corresponding to the first substream and so on.

Now, peer churn can happen in terms of peer arrivals and departures in both  $C_1$  and  $C_{1+}$ . Since the distribution graphs for the peers in  $C_{1+}$  are exactly as before, peer churn can also be handled similarly. However, for the peers in  $C_1$ , churn in  $C_{1+}$  translates as dynamics in the number of substream sources. In addition, they have to handle the peer churn happening within their cluster. The latter is handled as in the homogeneous scenario (section 4) since the distribution graphs of  $C_1$  have Properties 1 and 2 for

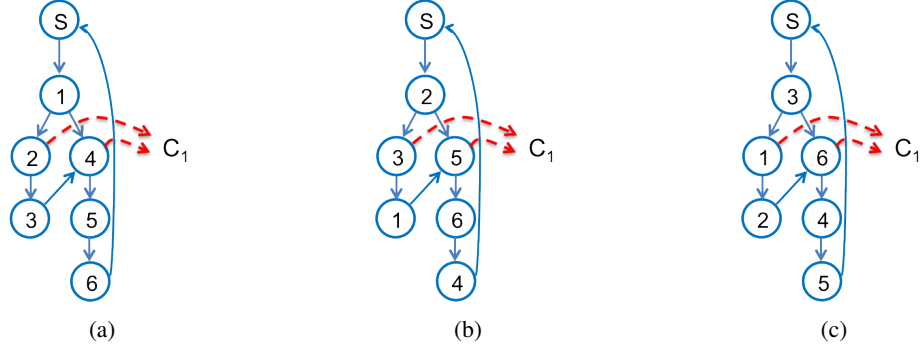


Figure 10. A cluster  $C_{1+}$  with 6 peers. The nodes with dotted-red edges act as source nodes for  $C_1$ .

small block departures (Proposition 3), while the connectivity property ensures that the peers continue to receive the stream even when some of the substream sources from  $C_{1+}$  leave the system.

As with churn management, balance has to be achieved in both the clusters. For the peers in  $C_{1+}$  using the algorithm of section 4 this is automatically guaranteed. However, for the nodes in  $C_1$ , the delay is minimized if the trees corresponding to each source in each substream are of similar size. Note that every root node of a substream in  $C_1$  can know the size of its tree since it receives edges from its end node and the end node of the previous tree. By exchange of this information among the root nodes, they can direct the source nodes to make connections such that the trees are of similar sizes. For example, in Figure 9(a) the end node 6 forwards the label information to the root nodes 1 and 7. Similarly node 12 forwards the label information to the two root nodes. As such, by taking the difference of the received label, the root nodes 1 and 7 can know the size of their respective subtrees as 6. The root nodes can also know the size of the neighboring tree by exchanging this information using the end nodes 6 and 12. As such, if there are  $k$  sources, each subtree root can know the size of its own subtree and the neighboring subtrees. The root nodes can then use this information to direct the source nodes for that substream to find new root nodes such that the  $k$  subtrees are approximately equal in size. Then, balance within the trees can be achieved as in the homogeneous case.

Thus, our algorithm can easily be extended to cover the “low-rate low-delay” regime of heterogeneous networks. Details and analysis are left to the full-paper.